Team-building with Answer Set Programming in the Gioia-Tauro Seaport

F. Ricca[1], G. Grasso[1,3], M. Alviano[1], M. Manna[1], V. Lio[2], S. Iiritano[2], N. Leone[1]

[1] *Dipartimento di Matematica, Università della Calabria, 87030 Rende, Italy*
[2] *Exeura s.r.l., Via Pedro Alvares Cabrai - C.da Lecco 87036 Rende (CS), Italy*
[3] *Computing Laboratory, University of Oxford, UK*
E-mail: {ricca,grasso,alviano,manna,leone}@mat.unical.it,
{vincenzino.lio,salvatore.iiritano}@exeura.it

submitted 24 July 2010; revised 4 Jan 2011; accepted 16 Jan 2011

Abstract

The seaport of Gioia Tauro is the largest transshipment terminal of the Mediterranean coast. A crucial management task for the companies operating in the seaport is team building: the problem of properly allocating the available personnel for serving the incoming ships. Teams have to be carefully arranged in order to meet several constraints, such as allocation of the employees with the appropriate skills, fair distribution of the working load, and turnover of the heavy/dangerous roles. This makes team building a hard and expensive task requiring several hours per day of manual preparation.

In this paper we present a system based on Answer Set Programming (ASP) for the automatic generation of the teams of employees in the seaport of Gioia Tauro. The system is currently exploited in the Gioia Tauro seaport by ICO BLG, a company specialized in automobile logistics.

KEYWORDS: Answer Set Programming, Declarative Problem Solving, Knowledge Management, Workforce Management, Artificial Intelligence.

1 Introduction

In the last few years, the need for knowledge-based technologies has emerged in several application areas. Industries are now looking for *semantic* instruments for knowledge management enabling complex domain-knowledge modeling and real-world problem solving. We report on a recent successful industrial application we developed in the area of Knowledge Management (KM), which is based on Answer Set Programming (ASP) (Gelfond and Lifschitz 1991). ASP is a fully-declarative language for Knowledge Representation and Reasoning (KRR), which has been developed in the field of logic programming and nonmonotonic reasoning. ASP has been already exploited for solving complex knowledge-based problems arising in Artificial Intelligence (Baral and Gelfond 2000; Balduccini et al. 2001; Baral and Uyan 2001; Friedrich and Ivanchenko 2008; Franconi et al. 2001; Gebser et al. 2007; Nogueira et al. 2001) as well as in Information Integration (Leone et al. 2005), and other areas of Knowledge Management (Baral 2003; Bardadym 1996; Grasso et al. 2009). In recent years,

the high knowledge-modeling power of this language has been made available to users, who need to represent and manipulate complex knowledge, by the development of some efficient ASP systems, such as DLV (Leone et al. 2006).

In this paper, we present a new system, based on the ASP solver DLV, which is currently exploited by the ICO BLG company in the international seaport of Gioia Tauro. The system allows for the automatic generation of teams of employees. Roughly, the problem we deal with is a *workforce management* problem (Naveh et al. 2007; Ernst et al. 2004; Tien and Kamiyama 1982; Lau 1996; Yang 1996), which amounts to computing a suitable allocation of the available personnel of the seaport such that cargo ships mooring in the port are properly handled. To accomplish this task several constraints have to be satisfied. An appropriate number of employees, providing several different skills, is required depending on the size and the load of cargo ships. Moreover, the way an employee is selected and the specific role she will play in the team (each employee is able to cover several roles according to her skills) are subject to many conditions (e.g., fair distribution of the working load, turnover of the heavy/dangerous roles, etc.). Up to now, the management of ICO BLG has been obliged to dedicate several hours per day to the difficult and delicate task of allocating teams of employees to incoming ships. Indeed, a bad allocation might cause delays or violations of the contract with shipping companies, with consequent pecuniary sanctions. Moreover, an unfair distribution of the workload can lead to problems with human resources management, ranging from: (i) the unavailability of crucial human resources -when highly-skilled employees are not allocated in the best way- (ii) to the loss of employee happiness -resulting from an unfair distribution of heavy/dangerous roles- that can directly affect production and performance. Thus, team building is definitively a crucial and difficult management task for ICO BLG, for which we developed an effective software solution.

The system described in this paper can either build new teams or automatically complete a partial allocation when the user manually pre-assigns some key employees. Moreover, it is able to verify whether a manually-composed team satisfies all the requirements, or not. It might be the case that there is no team satisfying all requirements, or that a team violates some condition due to user pre-assignments. In these cases, the system provides a suitable explanation of violations for allowing the user to take corrective actions. Further, the system can also be used by the managers of the company for performing simulations in order to estimate important financial aspects. For instance, by computing ahead the whole shift scheduling for the next month, it is possible to evaluate how much overtime will be needed and allow/deny leave requests. Notably, in this application, the domain knowledge is declaratively modeled by exploiting ASP and implemented by using the ASP system DLV. Desired allocations are computed by means of a set of suitably defined ASP programs.

The contribution of this paper is twofold: firstly, we report on a successful industrially-developed KM system which is currently employed in the important port of Gioia Tauro (the system is commercially-distributed by EXEURA s.r.l., a spin-off company of the University of Calabria); secondly, we describe how ASP can be profitably exploited for solving a specific KM problem. The work represents a practical

demonstration that ASP is well-suited for developing effective real-world KM systems.

In the literature there is a number of systems and methods for solving various variants of the workforce management problem (Naveh et al. 2007; Ernst et al. 2004; Tien and Kamiyama 1982; Lau 1996; Yang 1996). The available solutions can be classified in two main categories: (i) ad-hoc algorithms, and (ii) generic methods. The first category has the drawback of producing solutions that are very specific which cannot be easily adapted for solving a different variants of the same problem; in practice they are employed for solving problems with unique features (see e.g., Burke and Soubeiga ; Hultberg and Cardoso 1997). Conversely, our approach belongs to the second category where, historically, both Operational Research and Artificial Intelligence methods (such as Constraint Programming, Fuzzy logic, Genetic Algorithms, and Local Search) were employed (see e.g., Naveh et al. 2007; Ernst et al. 2004; Tien and Kamiyama 1982; Lau 1996; Yang 1996; Cerulli et al. 1992; Lesaint et al. 2003; Eitzen et al. 2004; Dechter 2004; Gresh et al. 2007; Bechtold et al. 1991; Alfares 2002; Billionnet 1999; Sun et al. 1998; Aickelin and Dowsland 2000; Wren and Wren 1995; Chiu et al. 2005; Rossi 2000; Al-Yakoob and Sherali 2007; Alba and Chicano 2007).[1] The solutions presented in this paper also demonstrate that ASP can be exploited on a par with other AI techniques for developing effective workforce management systems. The distinctive features of our approach are:

- the distinction of individual employees[2] by taking into account (i) specific employee-skills, (ii) current allocation history, and (iii) temporary unavailability (owing to illness or special permission);
- the satisfaction of practically-relevant cooperative work requirements, like the turnover of heavy/dangerous roles; and "double shift" allocations (where the same workers have to be employed in two adjacent shifts while respecting constraints);
- system interactivity: the user can manually modify the provided solutions, since the system is able to deal with partial allocations and provide explanations when constraints are violated by manual modification;
- high flexibility and maintainability: the system exploits a flexible, and easy-to-maintain core logic;
- user friendly graphical interface that makes the use of the underlying technologies transparent to the user.

An important lesson we learned in this project is that the high expressiveness of ASP language, which allows us to obtain an executable specification, is an important advantage that can lead to prefer ASP over other approaches in applications of this kind. Indeed, the expressiveness of ASP and its purely declarative nature

[1] For an in-depth comparison/characterization of solved problems and adopted methods we refer the reader to the following survey papers (Naveh et al. 2007; Ernst et al. 2004; Tien and Kamiyama 1982; Lau 1996; Yang 1996).

[2] A feature missing in most of the existent approaches, as observed by Naveh et al. (2007).

allowed us to refine and fine-tune both problem specifications and encodings together, while interacting with the stakeholders of the port. We could easily simulate specification changes and immediately show their effects to the customers. The possibility of modifying (simply by editing a text file) a complex reasoning task (e.g., by adding new constraints or by changing existing ones) in a few minutes and testing it "on-site" together with the customer is a great advantage of our approach, it was highly appreciated by our customers, and constituted a key factor for the success of our application. We believe that such an advantage could be exploited in several contexts.

The result is a system that perfectly matches the requirements of the customer by allowing both fast planning and fair employee allocation, while respecting the constraints. The management of ICO BLG can now produce an admissible personnel allocation in a few minutes: they spend less time and obtain a more effective planning of the resources.

The remainder of this paper is organized as follows: in the next section we give an overview of ASP; Section 3 provides a specification of the team building problem; while in Section 4 we give an ASP encoding of the problem; Section 5 describes the architecture of the system and its functionalities; Section 6 reports on the performances of the system on real-world data; Section 7 concludes the paper.

2 Answer Set Programming

In this section we recall syntax and semantics of Answer Set Programming (ASP). More specifically, we present an enriched language allowing for using set-oriented functions, also known as aggregate functions. For further background on the standard ASP language we refer to Baral (2003) and Gelfond and Leone (2002). For details on ASP with aggregate functions, instead, we refer to Faber et al. (2004) and Lee and Meng (2009).

2.1 Syntax

We assume sets of *variables*, *constants*, and *predicates* to be given. Similar to Prolog, we assume variables to be strings starting with uppercase letters and constants to be non-negative integers or strings starting with lowercase letters. Predicates are strings starting with lowercase letters. An *arity* (non-negative integer) is associated with each predicate. Moreover, the language allows for using built-in predicates (i.e., predicates with a fixed meaning) for the common arithmetic operations (i.e., $=$, $\leq$, $<$, $>$, $\geq$, $+$, etc.; usually written in infix notation).

Standard Atoms. A *term* is either a variable or a constant. A *standard atom* is an expression $\mathtt{p(t_1,\ldots,t_k)}$, where $\mathtt{p}$ is a *predicate* of arity k and $\mathtt{t_1,\ldots,t_k}$ are terms. If $\mathtt{t_1,\ldots,t_k}$ are constants, $\mathtt{p(t_1,\ldots,t_k)}$ is a *ground standard atom*.

Set Terms. A *set term* is either a symbolic set or a ground set. A *symbolic set* is a pair $\{Terms : Conj\}$, where *Terms* is a list of terms (variables or con-

stants) and *Conj* is a conjunction of standard atoms. Intuitively, a set term `X: a(X,Y), b(Y)` stands for the set of `X`-values making the conjunction `a(X,Y), b(Y)` true, namely $\{$`X` $\mid \exists$ `Y` *s.t.* `a(X,Y), b(Y)` *is true*$\}$. A *ground set* is a set of pairs of the form $\langle consts : conj \rangle$, where *consts* is a list of constants and *conj* is a conjunction of ground standard atoms.

Aggregate Functions. An *aggregate function* is of the form $f(S)$, where S is a set term and f is an *aggregate function symbol*. Intuitively, an aggregate function can be thought of as a (possibly partial) function, mapping multisets of constants to constants. Hereinafter, we will adopt the notation of the DLV system (Leone et al. 2006) for representing aggregate functions:

- `#min` (minimal term, undefined for empty set);
- `#max` (maximal term, undefined for empty set);
- `#count` (number of terms);
- `#sum` (sum of integers).

Aggregate Atoms and Literals. An *aggregate atom* is a structure of the form $f(S) \odot T$, where $f(S)$ is an aggregate function, $\odot \in \{<, \leq, >, \geq, =, \neq\}$ is a comparison operator, and T is a term (a variable or a constant). If T is a constant and S is a ground set term, $f(S) \odot T$ is a *ground aggregate atom*. A *literal* is either (i) a standard atom, or (ii) a standard atom preceded by the *negation as failure* symbol `not`, or (iii) an aggregate atom.

Programs. A *rule* r is a construct of the form

$$\alpha_1 \vee \cdots \vee \alpha_n \ :- \ \ell_1, \ \ldots, \ \ell_m.$$

where $\alpha_1, \ \ldots, \ \alpha_n$ are standard atoms, $\ell_1, \ \ldots, \ \ell_m$ are literals, $n \geq 1$ and $m \geq 0$. The disjunction $\alpha_1 \vee \cdots \vee \alpha_n$ is referred to as the *head* of r, while the conjunction $\ell_1, \ \ldots, \ \ell_m$ is the *body* of r. We denote by $H(r)$ the set of head atoms and by $B(r)$ the set of body literals. If $H(r)$ and all the literals in $B(r)$ are ground, r is a *ground rule*. A ground rule r is a *fact* if both $B(r) = \emptyset$ and $|H(r)| = 1$. A *program* $\mathcal{P}$ is a set of rules; if all rules in $\mathcal{P}$ are ground, $\mathcal{P}$ is a ground program.

Safety. A *local* variable of a rule r is a variable appearing solely in set terms of r; a variable of r that is not local is *global*. A rule r is *safe* if both the following conditions hold: (i) each global variable appears in some positive standard literal of $B(r)$; (ii) each local variable appearing in a set term $\{Terms : \ Conj\}$ also appears in *Conj*. A program is safe if all its rules are safe. In the following, we assume that programs are safe.

Example 1
Consider the following rules:

```
c(X) :- q(X,Y,V), #max{Z: a(Z), b(Z,V)} > Y.
c(X) :- q(X,Y,V), #sum{Z: a(X), b(X,S)} > Y.
c(X) :- q(X,Y,V), #min{Z: a(Z), b(Z,V)} > T.
```

The first rule is safe, while the second is not because the local variable `Z` violates condition (ii). The third rule is not safe because the global variable `T` violates condition (i). ∎

2.2 Answer Set Semantics

Semantics of an ASP program is given by the set of its stable models, defined in this section.

Universe and Base. Given an ASP program $\mathcal{P}$, the *universe* of $\mathcal{P}$, denoted by $U_\mathcal{P}$, is the set of constants appearing in $\mathcal{P}$.[3] The *base* of $\mathcal{P}$, denoted by $B_\mathcal{P}$, is the set of standard atoms constructible from predicates of $\mathcal{P}$ with constants in $U_\mathcal{P}$.

Instantiation. A *substitution* is a mapping from a set of variables to $U_\mathcal{P}$. Given a substitution σ and an ASP object *obj* (atom, rule, set term, etc.), we denote by $obj\,\sigma$ the object obtained by replacing each variable `X` in *obj* by σ(`X`). A substitution from the set of global variables of a rule r (to $U_\mathcal{P}$) is a *global substitution* for r; a substitution from the set of local variables of a set term S (to $U_\mathcal{P}$) is a *local substitution* for S. Given a set term without global variables $S = \{\,Terms : Conj\,\}$, the *instantiation of* S is the following ground set term:

$$inst(S) = \{\langle Terms\,\sigma :\ Conj\,\sigma\rangle \mid \sigma \text{ is a local substitution for } S\}.$$

A *ground instance* of a rule r is obtained in two steps: first, a global substitution σ for r is applied; then, every set term S in $r\sigma$ is replaced by its instantiation $inst(S)$. The instantiation $Ground(\mathcal{P})$ of a program $\mathcal{P}$ is the set of instances of all rules in $\mathcal{P}$.

Example 2
Consider the following program $\mathcal{P}_1$:

```
a(1) v b(2,2).
a(2) v b(2,1).
c(X) :- a(X), #sum{Y: b(X,Y)} ≥ 2.
```

The instantiation $Ground(\mathcal{P}_1)$ of $\mathcal{P}_1$ is the following program:

```
a(1) v b(2,2).
a(2) v b(2,1).
c(1) :- a(1), #sum{⟨1: b(1,1)⟩, ⟨2: b(1,2)⟩} ≥ 2.
c(2) :- a(2), #sum{⟨1: b(2,1)⟩, ⟨2: b(2,2)⟩} ≥ 2.
```

∎

Interpretations. An *interpretation* I for an ASP program $\mathcal{P}$ is a subset of $B_\mathcal{P}$. A standard ground atom α is true with respect to I if $\alpha \in I$; otherwise, α is false with respect to I. A negative standard ground literal `not` α is true with respect to I if $\alpha \notin I$; otherwise, `not` α is true with respect to I.

An interpretation I also provides a meaning to set terms, aggregate functions

[3] If a program $\mathcal{P}$ has no constants, an arbitrary constant symbols ξ is introduced.

and aggregate literals, namely a multiset, a value, and a truth value, respectively. The evaluation $I(S)$ of a set term S with respect to I is the multiset $I(S)$ defined as follows: Let $S^I = \{\langle t_1, ..., t_k\rangle \mid \langle t_1, ..., t_k : Conj\rangle \in S$ and all the atoms in $Conj$ are true with respect to $I\}$; then, $I(S)$ is the multiset obtained as the projection of the tuples of S_I on their first constant, that is, $I(S) = [t_1 \mid \langle t_1, ..., t_n\rangle \in S^I]$. The evaluation $I(f(S))$ of an aggregate function $f(S)$ with respect to I is the result of the application of f on $I(S)$. If the multiset $I(S)$ is not in the domain of f, $I(f(S)) = \bot$ (where $\bot$ is a fixed symbol not occurring in $\mathcal{P}$). A ground aggregate atom $f(S) \odot k$ is true with respect to I if both $I(f(S)) \neq \bot$ and $I(f(S)) \odot k$ hold; otherwise, $f(S) \odot k$ is false.

Models. Given an interpretation I, a rule r is *satisfied* with respect to I if some head atom is true with respect to I whenever all body literals are true with respect to I. An interpretation M is a *model* of an ASP program $\mathcal{P}$ if all the rules $r \in Ground(\mathcal{P})$ are satisfied with respect to M. A model M for $\mathcal{P}$ is (subset) minimal if no model N for $\mathcal{P}$ exists such that $N \subsetneq M$.[4]

Answer Sets. We now recall the generalization of the Gelfond-Lifschitz transformation and answer sets for programs with aggregates from Faber et al. (2004): Let $\mathcal{P}$ be a ground program, I be an interpretation, and $\mathcal{P}^I$ denote the transformed program obtained from $\mathcal{P}$ by deleting all rules in which a body literal is false with respect to I. An interpretation M is an answer set of the program $\mathcal{P}$ if it is a minimal model of $Ground(\mathcal{P})^M$.

Example 3
Consider two interpretations $I_1 = \{\texttt{a(0)}\}$ and $I_2 = \{\texttt{b(0)}\}$ for the next programs:

$\mathcal{P}_2$ = {a(0) :- #count{X: b(X)} > 0.}
$\mathcal{P}_3$ = {a(0) :- #count{X: b(X)} ≤ 0.}

Hence, we obtain the following transformed programs:

$Ground(\mathcal{P}_2)^{I_1} = \emptyset$
$Ground(\mathcal{P}_2)^{I_2} = Ground(\mathcal{P}_2) =$ {a(0) :- #count{⟨0: b(0)⟩} > 0.}
$Ground(\mathcal{P}_3)^{I_1} = Ground(\mathcal{P}_3) =$ {a(0) :- #count{⟨0: b(0)⟩} ≤ 0.}
$Ground(\mathcal{P}_3)^{I_2} = \emptyset$

Hence, neither I_1 nor I_2 are answer sets of $\mathcal{P}_2$, while I_1 is an answer set of $\mathcal{P}_3$ because $\emptyset$ is not a model of $Ground(\mathcal{P}_3)^{I_1}$. On the other hand, I_2 is not an answer set of $\mathcal{P}_3$ because $\emptyset$ is a model of $Ground(\mathcal{P}_3)^{I_2}$. ∎

Note that the syntax of the language does not explicitly allow rules without head atoms, called constraints. However, constraints can be simulated by using a new symbol and negation. For example, a constraint of the form $:\!-\ \ell_1, \ \ldots, \ \ell_m$ can be replaced by a rule of the form $co :\!-\ \ell_1, \ \ldots, \ \ell_m,$ `not` co, where co is a symbol not occurring in $\mathcal{P}$.

[4] Note that an answer set M of $\mathcal{P}$ is also a model of $\mathcal{P}$. Indeed, $Ground(\mathcal{P})^M \subseteq Ground(\mathcal{P})$ and rules in $Ground(\mathcal{P}) \setminus Ground(\mathcal{P})^M$ are satisfied with respect to M by definition of reduct.

3 Problem Motivation and Specification

The seaport of Gioia Tauro[5] is the largest transshipment terminal of the Mediterranean coast. Most of the transported goods reach and leave this port by ship, and only a minimal part of them (about 5%) is transferred by train or truck. Historically, container transshipment has been the main activity of the port (related problems were the subject of extensive research; see e.g., Vacca et al. 2007), while recently Gioia Tauro has become also an automobile hub. Automobile logistics is carried out by the ICO BLG company[6] on an area of 320,000 m^2 corresponding to a warehouse capacity for about 15,000 vehicles. In the last few years, cargo traffic has increased impressively in Gioia Tauro: every day several vessels of different capacities moor in the port carrying their load of new cars and trucks. The vehicles they transport might have to be handled, warehoused, processed (if technically necessary) and finally delivered to their final destination. Depending on the specific processing needs, the load of a mooring ship might be subject to different activities, each of which requires the allocation of a number of specifically-skilled employees. For instance, a ship might be partially or fully loaded/unloaded, and transported vehicles identified, quality-checked and moved to/from the port yard. Moreover, cars (possibly-damaged during transportation) might undergo an additional repairing process. The port is, in fact, equipped with a centre able to repair both minor damage to car bodywork and to perform pre-delivery services such as polishing, washing, battery testing, etc.

The time required for processing ships is set up by contract. For this reason, a critical management goal is to serve arriving boats promptly because any delay might cause financial penalties to the company. In order to accomplish that task, several teams of employees have to be arranged and scheduled to work simultaneously on these ships. The selection of the employees and the assignment of roles is subject to many conditions. Some constraints are imposed by the employees skills and availability, other by contract (in general identified as legal conditions). For example, each team usually includes at least one "quality-checker", and enrolls a sufficient number of drivers for loading/unloading cars in time. Moreover, each employee cannot work more than 36 hours (plus max 12 hours of overtime) per week and, importantly, heavy/dangerous roles must be turned over. Note that some tasks require that employees work in "bad conditions"; for instance, the employees performing operations in the hold (while cars are driven in/out of a ship) are persistently exposed to a high concentration of pollution. Hence, daily assignments to such heavy roles must be turned over among the personnel, so that hard and easy tasks are shared among employees. Moreover, a fair distribution of work has to be obtained by ensuring that all the employees are enrolled for about the same number of hours in a week. In this way, both the best possible working conditions for the employees and the processing of all incoming boats are guaranteed.

Data regarding arriving/departing ships, such as date and time, number and kind

[5] See `http://www.portodigioiatauro.it`
[6] ICO BLG is a subsidiary of the BLG Logistics Group — `http://www.blg.de`

of transported vehicles, are usually known by the managers (at least) one day in advance. Given these data, the management easily identifies the team composition characteristics needed for processing a given ship (i.e., how many employees are needed, what skills the workers must have, how the working time is divided in shifts, etc.). According to ICO BLG terminology, we say that a meta-plan is a specification of the needed skills and number of employees for one shift. (A meta-plan usually refers to a single ship.) However, the subsequent task of assigning the available employees to shifts and roles (remember that each employee is able to cover several roles according to her skills), in such a way that all the requirements are satisfied, is definitively the hardest part of the whole rostering process.

The ICO BLG managers were used to find out a "hand-made" solution. Of course, this technique was far from being effective and efficient. Indeed, arranging teams required many hours, often resulting in solutions not fulfilling several constraints. The impossibility of allocating teams to incoming ships might cause delays and violations of the contract with employees or with shipping companies. As a consequence, *team building is definitively a crucial management task* for ICO BLG.

In the following, all the requirements to be fulfilled are specified in more detail.

3.1 Team Requirements and Available Personnel

Information regarding mooring ships, such as arrival and departure time as well as their processing needs (load/unload, size of the load, etc.), is known one day in advance, and the rostering process is always carried out by considering one day at time. Moreover, working time of a day is divided in shifts lasting 6-12 hours each. Given this information, the managers can easily produce a specification of the resources needed during the working shifts for dealing with the incoming ships. In particular, the following information is specified: the shift data (date-time identifier and duration), the set of skills to be covered, and the number of needed employees per skill. This is the main input of our team building system because the goal is to select the right workers among those which are available for fulfilling resource requests. However, the availability of a specific employee for a given shift depends on several conditions, some of which are stated by the employees contract, some by other circumstances.

Concerning legal requirements, we already mentioned that an employee cannot exceeds a certain amount of working hours (including overtime) per day and per week. These values are given as input parameters within our system, together with the allocation history recording the number of elapsed working hours for each employee. A further condition imposed by the contract regards night-shifts. It states that an employee who has worked during a night-shift must have a mandatory rest time before being allocated once more. In addition, an employee can be assigned to a single vessel in a given shift, but she can be moved to a different vessel in a subsequent shift. An employee cannot change role in a shift, but she can play different roles in different shifts. Finally, an employee is not *available* if either she is in vacation, or a specific management decision requires that she must not be a member of the team.

3.2 Turnover and Fairness Requirements

As previously pointed out, to obtain the best possible working conditions for the employees, heavy/dangerous roles have to be turned over. More in detail, a "round-robin like" turnover policy should be applied for avoiding that the same group of employees is mostly allocated to heavy/dangerous roles. Moreover, a fair distribution of the workload has to be ensured in such a way that all the employees have to be allocated for about the same amount of weekly hours. This requirement concerns the way the workload is spread over employees within a week. As an example, suppose that the workload expected for a given week is relatively low. In this case, the possible solutions range from the exploitation of the same few employees (possibly up to their weekly limit of hours) and the distribution of the few shifts to distinct available employees. Clearly, the last scenario is the fairest and, in general, it should be avoided that a group of employees works less (or is unemployed) in a week, whereas some other group is overloaded. Basically, the larger admissible gap of worked hours among employees has to be maintained below a threshold (which can be set as a parameter in our system) of usually 8 hours.

3.3 Crucial Roles

The employees of ICO BLG, in general, can play different roles in a team, depending on their skills. For instance, the same employee can be enrolled in a team as a driver (her role is to drive cars in/out of the boats) or as a quality-checker, in case she is skilled on both activities. Clearly, there are a few employees skilled in critical roles (such as quality-checkers), while almost all employees are skilled for easy roles (e.g., all employees are drivers). If employees skilled in critical roles are not employed carefully, it might be the case that a team satisfying all the requirements cannot be built. Indeed, if these few employees are assigned to other (more common) roles during the first days of the week, it might happen that they become unavailable on the last days of the week (e.g., they may have already reached 36 working hours). In order to avoid these disadvantageous situations, employees having crucial roles have to be preserved if possible.

3.4 Double Shift

A very specific requirement of our team building problem is the possibility of specifying *double shifts*. So far, we have just considered (simple) shifts, where employees starts and finish working performing the same role. Instead, a double shift can be thought of as two consecutive (simple) shifts where the same employees are assigned to possibly different roles.[7] More in detail, let n_1 and n_2 be the number of employees required for two shift s_1 and s_2, respectively; a double shift has the following property: if s_2 does not require a higher number of workers than s_1 (i.e., $n_2 \leq n_1$), then only employees already working in s_1 have to be used in s_2; otherwise ($n_2 > n_1$),

[7] Note that this requirement cannot be enforced by considering single shifts only.

all the employees working in s_1 have to be allocated in s_2 and the team has to be (possibly) augmented by taking additional employees. Moreover, when moving from s_1 to s_2, roles must be turned over with respect to the assignment of s_1.

4 ASP-based Encoding

This section describes the ASP program which solves the team building problem specified in the previous section. First, the input data is described; then, the ASP rules computing teams of employees are described.

4.1 Input Specification

The input of the process is specified by means of the predicates described in this section. The association among employees and skills is expressed by instances of *hasSkill(employee, skill)*. Instances of *absent(employee, shift)* are used for storing information regarding employees being not available on a given shift because not present (e.g., for day off, for disciplinary decision, or for resting after a night-shift). In addition, instances of *manuallyExcluded(employee, shift)* are used for modeling employees who do not have to be considered according to some management decision (to allow manual intervention on the allocation of specific employees). Shifts are modeled by instances of *shift(shiftId, duration)*, reporting the duration of each shift. Moreover, the temporal sequence of shifts is represented by instances of *previousShift(shift, shift′)*, where *shift′* immediately precede *shift*. Resource requirements are represented by instances of *neededEmployees(shift, skill, n)*, reporting the number n of needed employees for each shift and skill.

Statistics concerning past allocations are modeled by instances of *workedWeeklyHours(employee, hours)* and *workedDailyHours(employee, hours)*, modeling worked hours for each employee up to the current allocation. Similarly, instances of *workedWeekOvertimeHours(employee, hours)* are used for modeling the count of overtime for each employee up to the current allocation. Instances of these predicates are produced by querying an internal database that is not described here to simplify the presentation. Few additional parameters of the process are given by means of the following predicates: an instance of *dailyHours(hours)* for specifying the maximum numbers of working hours allowed per day; an instance of *weekHours(hours)* for setting the maximum numbers of working hours allowed per week; an instance of *weekOvertime(hours)* specifying how many overtime hours per week an employee is allowed to work.

The predicates described up to now constitute the input of the subprogram reported in Section 4.2 and used for effectively produce all candidate solutions for the team building problem. The input in enriched by predicates allowing for discarding candidate solutions violating preference requirements on heavy/dangerous (Section 4.3) or crucial roles (Section 4.4). In particular, heavy/dangerous roles are specified by instances of *heavyRole(skill)* and, for each employee and skill, the last allocation date of that employee on that skill is given by an instance of *lastAllocation(employee, skill, lastTime)*. Moreover, the maximum admissible difference of

working hours among all the employees is given by an instance of *fairGap(hours)*. Concerning crucial skills, they are identified by instances of *crucialRole(skill)*. Moreover, the input contains an instance of *hasCrucial(employee, n)* for each employee, specifying the number n of her crucial skills.

Finally, double shifts are modeled by instances of *isDouble(shift$_1$, shift$_2$)*, where *shift$_1$* and *shift$_2$* are consecutive (simple) shifts, and the number of employees required by *shift$_1$* is less than or equal to the one required by *shift$_2$*. These facts are used by the subprogram reported in Section 4.5.

4.2 Team Requirements and Available Personnel

The following rule identifies the set of employees who can be assigned to a specific role in each shift:

```
canBeAssigned(Em,Sh,Sk) :- hasSkill(Em,Sk), neededEmployees(Sh,Sk,_),
   not absent(Em,Sh), not manuallyExcluded(Em,Sh),
   not exceedTimeLimit(Em,Sh).
```

In detail, an employee `Em` can be assigned to a shift `Sh` for a role `Sk` if the following conditions are satisfied: (i) she has the skill `Sk`; (ii) `Sh` needs employees with skill `Sk`; (iii) she is not absent (iv) she is not excluded by some managers decision; and (v) she does not exceed time limits. Predicate `exceedTimeLimit` is intentionally defined as follows:

```
exceedTimeLimit(Em,Sh) :- shift(Sh,D),
   workedWeeklyHours(Em,Wh), weekHours(Hmax), D + Wh  > Hmax.

exceedTimeLimit(Em,Sh) :- shift(Sh,D), dailyHours(Hmax),
   workedDailyHours(Em,Wh), D + Wh  > Hmax.

exceedTimeLimit(Em,Sh) :- shift(Sh,D), weekOvertime(Hmax),
   workedWeekOvertimeHours(Em,Wh), D + Wh  > Hmax.
```

In particular, the first rule can be read as follows: "if a shift `Sh` has duration `D` and, for a given employee `Em`, the sum of `D` with `Wh` (the hours worked by `Em` up to `Sh`) is greater than `Hmax` (the maximum allowed hours in a week), then `Em` exceeds the time limit when associated to `Sh`". The remaining two rules can be interpreted in a similar way if daily worked hours and overtime are considered instead.

Next, according to the guess&check programming methodology (Eiter et al. 2000; Leone et al. 2006), the following disjunctive rule guesses the selection of employees that can be assigned to the shifts in the appropriate roles:

```
assign(Em,Sh,Sk) v nAssign(Em,Sh,Sk) :- canBeAssigned(Em,Sh,Sk).
```

This rule can be read as follows: "if an employee `Em` matches the skill requirements for a shift `Sh`, then `Em` can be assigned to `Sh`, or not". Assignments violating team requirements are filtered out by the following constraints:

```
:- neededEmployees(Sh,Sk,EmpNum), #count{Em: assign(Em,Sh,Sk)} ≠ EmpNum.

:- assign(Em,Sh,Sk1), assign(Em,Sh,Sk2), Sk1 ≠ Sk2.
```

```
:- assign(Em,Sh1,_), assign(Em,Sh2,_), Sh1 ≠ Sh2.                    (*)
```

In particular, the first constraint discards assignments with a wrong number of employees per required skill (in other words, an assignment is discarded if the total count of employees assigned to a skill does not match requirements). On the other hand, the second and third constraints avoid that the same employee covers two roles or two shifts, respectively.[8]

4.3 Turnover and Fairness Requirements

A careful assignment to heavy/dangerous roles is probably one of the most important requirements for ensuring the best possible working conditions to employees. We achieved this behavior by employing the following rule:

```
prefByTurnover(Em1,Em2,Sh,Sk) :- heavyRole(Sk),
   canBeAssigned(Em1,Sh,Sk), canBeAssigned(Em2,Sh,Sk),
   lastAllocation(Em1,Sk,Date1), lastAllocation(Em2,Sk,Date2), Date1 < Date2.
```

According to this rule, if two employees $\texttt{Em}_1$ and $\texttt{Em}_2$ are both skilled on a heavy role `Sk`, and $\texttt{Em}_2$ has performed `Sk` more recently than $\texttt{Em}_1$, then $\texttt{Em}_1$ is preferred to $\texttt{Em}_2$. Therefore, for each shift and heavy role, predicate `prefByTurnover` states a priority ordering among employees, and the following constraint implements rotation of roles by forbidding assignments whenever turnover preferences are not respected:

```
:- prefByTurnover(Em1,Em2,Sh,Sk), assign(Em2,Sh,Sk), not assign(Em1,Sh,Sk).
```

In particular, for a given shift, the assignment of an employee $\texttt{Em}_2$ on a role `Sk` is forbidden when there is an unassigned employee $\texttt{Em}_1$ having higher priority than $\texttt{Em}_2$ on `Sk`. Clearly, this implies that admissible solutions are only those that actually ensure an effective turnover, applying a "round-robin" policy among workers. Following a similar approach, we can impose the fairness requirement for spreading the workload among employees. The next rule sets allocation priorities among available workers:

```
prefByFairness(Em1,Em2,Sh,Sk) :- fairGap(MaxGap),
   workedWeeklyHours(Em1,Wh1), workedWeeklyHours(Em2,Wh2),
   canBeAssigned(Em1,Sh,Sk), canBeAssigned(Em2,Sh,Sk), Wh1 + MaxGap < Wh2.
```

In detail, if the gap on weekly worked hours between employees $\texttt{Em}_2$ and $\texttt{Em}_1$ exceeds the maximum amount allowed, then $\texttt{Em}_1$ has higher priority than $\texttt{Em}_2$.

Solutions violating fairness preferences are discarded by the constraint

```
:- prefByFairness(Em1,Em2,Sh,Sk), assign(Em2,Sh,Sk), not assign(Em1,Sh,Sk).
```

[8] Recall that a single day is considered during an execution.

4.4 Crucial Roles

In order to preserve crucial roles, the assignment of employees qualified for few crucial skills has to be preferred. Note that employees having many crucial skills may act as "wildcards" and should be employed with parsimony. This aspect is encoded by the rule

```
prefByCrucial(Em1,Em2,Sh,Sk) :- hasCrucial(Em1,N1), hasCrucial(Em2,N2),
   canBeAssigned(Em1,Sh,Sk), canBeAssigned(Em2,Sh,Sk), N1 < N2.
```

modeling a priority order that depends on the number of crucial skills of employees. Finally, assignments not respecting preferences on crucial roles are discarded by the following constraint:

```
:- prefByCrucial(Em1,Em2,Sh,Sk), assign(Em2,Sh,Sk), not assign(Em1,Sh,Sk).
```

4.5 Double Shifts

A double shift comprises two simple shifts in which an employee is possibly enrolled in different roles (one for each component). However, constraint `(*)` in Section 4.2 prevents to enroll an employee in different roles in a run. Thus, for allowing the association of the same employees in a double shift, constraint `(*)` is modified as follows:

```
:- assign(Em,Sh1,_), assigned(Em,Sh2,_), Sh1 ≠ Sh2, not isDouble(Sh1,Sh2).
```

Moreover, recall that an instance of $\texttt{isDouble(Sh}_{small}\texttt{,Sh}_{big}\texttt{)}$ is in the input whenever $\texttt{Sh}_{small}$ and $\texttt{Sh}_{big}$ constitute a double shift, and the *number of employees* required by $\texttt{Sh}_{small}$ is less than or equal to the one required by $\texttt{Sh}_{big}$. Thus, the rules

```
:- isDouble(Sh_small,Sh_big), assigned(Em,Sh_small), not assigned(Em,Sh_big).

assigned(Em,Sh) :- assign(Em,Sh,_).
```

enforce that all employees assigned to $\texttt{Sh}_{small}$ must be assigned to $\texttt{Sh}_{big}$ as well. Finally, for respecting bounds on weekly, daily, and overtime working hours, the following constraints are added:

```
:- isDouble(Sh1,Sh2), assigned(Em,Sh1), assigned(Em,Sh2)
   shift(Sh1,D1), shift(Sh2,D2), workedWeeklyHours(Em,Wh),
   weekHours(Hmax), Wh + D1 + D2  > Hmax.

:- isDouble(Sh1,Sh2), assigned(Em,Sh1), assigned(Em,Sh2)
   shift(Sh1,D1), shift(Sh2,D2), workedDailyHours(Em,Wh),
   weekHours(Hmax), Wh + D1 + D2  > Hmax.

:- isDouble(Sh1,Sh2), assigned(Em,Sh1), assigned(Em,Sh2)
   shift(Sh1,D1), shift(Sh2,D2), workedWeekOvertimeHours(Em,Wh),
   weekHours(Hmax), Wh + D1 + D2  > Hmax.
```

In particular, these constraints discard assignments in which an employee violates time constraints when enrolled in both components of a double shift.

4.6 Dealing with Inconsistency

The ASP program described so far provides admissible solutions of the team building problem where all the requirements are satisfied. It is easy to see that the combined action of different constraints may cause situations where there is no admissible solution. As an example, suppose we want to compose a team of two people covering two different roles, where skill s_1 and s_2 are required. Suppose also that s_1 is a skill subject to turnover, and e_1 is the only employee that can be assigned to s_1 for respecting this requirement; moreover, all the employees but e_1 have worked for 20 hours so far, while e_1 has just worked for 10 hours (note that such a situation is not unusual, because e_1 may have worked during night in its last assignment). According to the fairness requirement, no one should be preferred to e_1 for covering s_2. This leads to a conflicting scenario where e_1 is the only possible choice for both s_1 and s_2, but the same employee cannot cover two distinct roles in the same team. Similar conflicting situations can occur in more complex settings where similar "loops" might happen.

Note that, in a real world scenario, it is often the case that all the constraints cannot be simultaneously satisfied. In order to deal with such a problem, together with ICO BLG managers, we devised a strategy applying constraints in a prioritized way. In particular, the managers of the company specified that the constraints have to be applied in the following importance order: (1) turnover, (2) fairness, and (3) crucial roles conditions (where the application of the leftmost has higher precedence). The idea is to avoid the application of more than one constraint on the same pair of employees. For instance, if an employee $\texttt{Em}_1$ is preferred to $\texttt{Em}_2$ according to the turnover, then fairness and crucial roles preferences must not be applied on the same workers. Thus, even if $\texttt{Em}_2$ would be preferable to $\texttt{Em}_1$ because of the fairness constraint, still $\texttt{Em}_1$ should be possibly taken. In order to implement this specification, we devised a prioritized variant of the above-mentioned constraints:

```
:- prefByCrucial(Em1,Em2,Sh,Sk), assign(Em2,Sh,Sk),
   not assign(Em1,Sh,Sk), not prefByTurnover(Em1,Em2,Sh,Sk),
   not prefByFairness(Em1,Em2,Sh,Sk).

:- prefByFairness(Em1,Em2,Sh,Sk), assign(Em2,Sh,Sk),
   not assign(Em1,Sh,Sk), not prefByTurnover(Em1,Em2,Sh,Sk).
```

The first constraint applies crucial role priority among two employees only if both turnover of roles and fairness requirements do not apply on the same employees. In a similar way, the second constraint enforces fairness only if the turnover constraint does not apply. Note that the constraint implementing turnover requirement did not need any reformulation. Indeed, they have the highest priority and, thus, have to be applied in any case.

Within our system, the encoding modified with constraint priority is run automatically if no answer set is produced by the original encoding. In this way, the system can find "acceptable" solutions by ignoring constraints that are in conflict with ones of higher priority. The system alerts the user when such a case occurs.

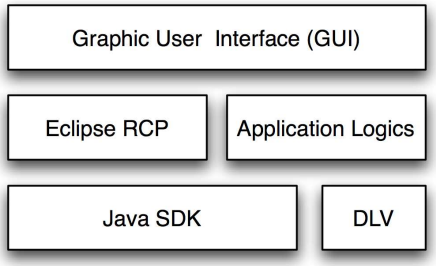

Fig. 1. System Architecture

4.7 Team Checking

Upon an explicit customer request, our system also allows the user to modify a computed team (partially or completely). Whenever a team is manually modified, the system checks whether it violates some constraints and informs the user. The checking is carried out by running the same program above, where the "guessing" rule is replaced by a set of facts for the "assign" predicate for specifying the team.

In case of inconsistency, one might be interested in finding the causes of the conflict. To this end, a slightly modified version of the same program can be exploited for detecting violated constraints, where specifically conceived atoms are introduced in the head of the constraints modeling preference conditions. For example, the constraint requiring the turnover of heavy/dangerous roles is replaced by the following rule:

```
violatedTurnover(Em1,Em2,Sh,Sk) :- prefByTurnover(Em1,Em2,Sh,Sk),
   assign(Em2,Sh,Sk), not assign(Em1,Sh,Sk).
```

An instance of `violatedTurnover(`$\texttt{Em}_1$`,`$\texttt{Em}_2$`,Sh,Sk)` is derived whenever employees $\texttt{Em}_1$ and $\texttt{Em}_2$ violate the turnover constraint for shift `Sh` and skill `Sk`. Clearly, analogous modifications can be applied to all the problem constraints. In this way, suitable explanation of violations can be provided so that the user can take corrective actions.

5 System Architecture and Usage

The architecture of the team-building system is depicted in Figure 1. The system integrates the ASP system DLV (Leone et al. 2006) and features a Graphical User Interface (GUI) developed in Java. In particular, the GUI is based on the Rich Client Platform (RCP) technology exploiting an application logics (also developed in Java) which embeds OntoDLV (Ricca et al. 2009) (an ontology management and reasoning system based on DLV) for implementing both the reasoning services and data-storage features.

The GUI combines in a single customizable frame all the controls (see Figure 2). In particular, a tree-shaped calendar (displayed on the left) allows for browsing and scheduling working activities. Meta-plans specifications, usually identified by the name of the handled cargo boats (e.g., Velasquez, Autoroute), are the leaves of the

tree; they can be added or removed by right-clicking on their name and selecting the proper command from a context-menu. Meta-plans information (ship arrival and departure date, available processing time and requested skills) is displayed in and editable through the "Logistics" panel. Below, the "Inclusion" and "Exclusion" panels allow pre-assigning (or excluding) specific employees from the team. To run the team builder engine, the user select a meta-plan (from the tree) and chooses the "run" item from a context-menu (activated by right-clicking the selection).[9] Once a meta-plan is run, input information and personnel statistics are fed into the DLV system and the result is displayed on the top-right panel ("Team Properties"). The computed team can be also modified manually, and the system is able to verify if the manually-modified team still satisfies the constraints. In case of errors, causes are outlined and suggestions for fixing a problem proposed. The interface gives full control on the status of all the seaport-staff: available/unavailable personnel is listed on the bottom-right panel, and the allocation statistics are reported in the bottom panel.

6 Some Experiments on Real-world Data

To provide a concrete image on the behavior of the system, we report on a use case performed on real-world data. In particular, we have simulated the employee-allocation on archive data provided by ICO BLG, regarding 130 employees and the activity of one month. We condidered four allocation activities: (r_1) one shift, (r_2) a daily allocation, (r_3) a weekly allocation, and (r_4) a monthly allocation. The system was installed in the ICO BLG workstation featuring an Intel Core 2 Duo CPU P8600 machine clocked at 2.40 GHz with 4 GB of RAM running Microsoft Windows XP and Java version 1.6.0_14. The runtime for accomplishing the simulation of (r_1), (r_2), (r_3) and (r_4) was: 4.2 seconds, 25.3 seconds, 128.7 seconds, and 490.6 seconds, respectively. Basically, in a few seconds the system is able to generate a shift plan, and in some minutes one can simulate a complete allocation covering an entire month (this feature can be used for estimating both long-range and mid-range employees allocation needs).

In order to assess the quality of the obtained plans, we compared the results of our system with the manually-prepared solution provided by the ICO BLG managers. The results confirm the higher quality of the solutions provided by our system; indeed, the number of constraints violations results dramatically reduced when the system-produced allocations are compared with hand-made ones. Further, the solutions provided by the system would have caused a decrease of about 20% in the amount of overtime needed. Those results confirmed the practical effectiveness of the ASP-based approach, and definitely convinced the ICO BLG management to adopt our system for workforce management.

[9] Note that multiple meta-plans can be executed by selecting several items from the tree.

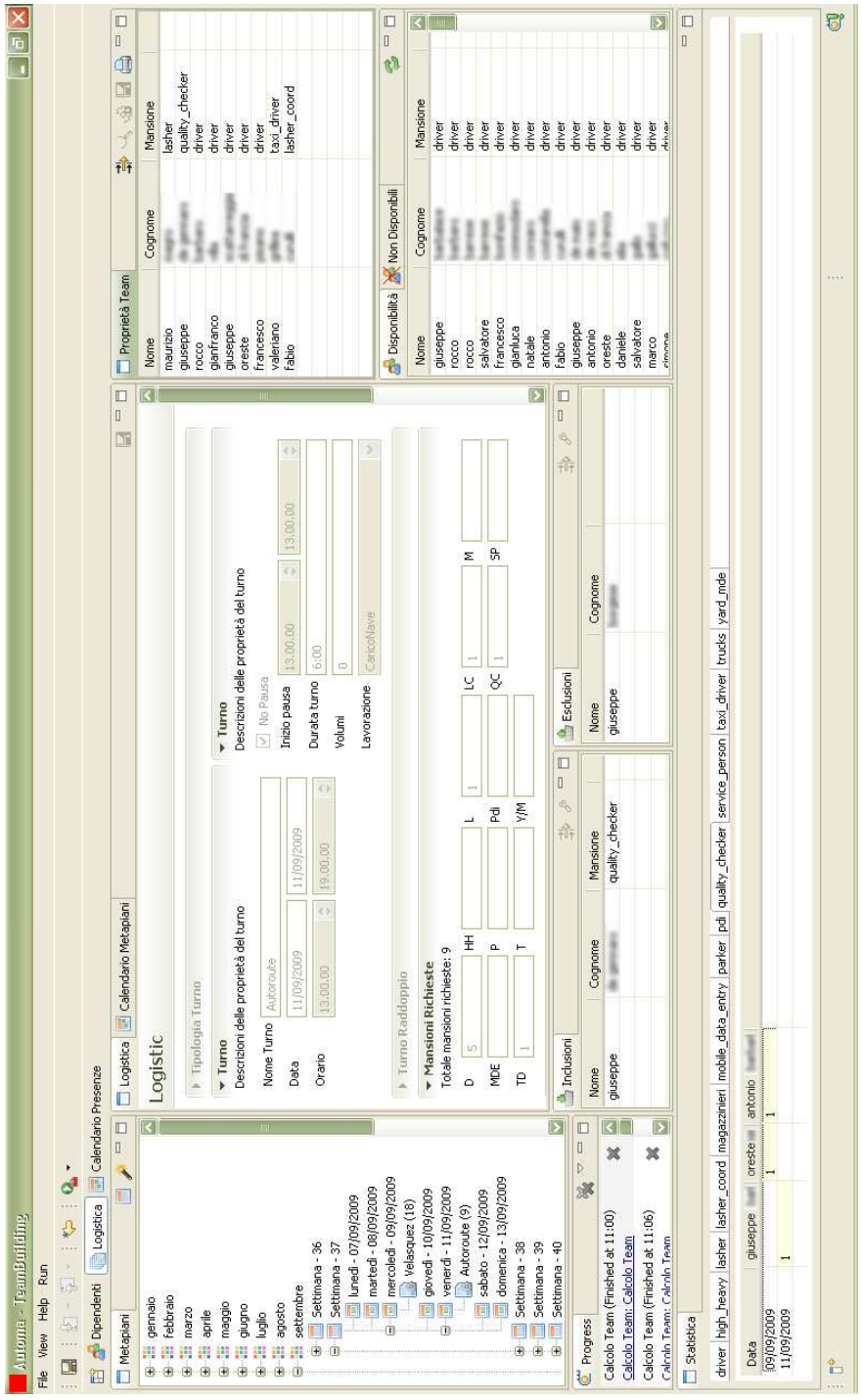

Fig. 2. The Team-builder Graphic User Interface

7 Conclusion

In this paper we have presented a team building system based on Answer Set Programming. The system features a graphical user interface that gives full control on the status of the seaport-staff and allows for transparently running the DLV system for computing, checking and completing suitable teams of employees respecting the given constraints. The system has been developed side by side with the personnel of ICO BLG and is currently employed by the management staff of that company. The practical effectiveness of the proposed solution is confirmed by the on-the-field usage impressions reported by the ICO BLG management members: "the system is able to obtain more reliable results when compared to manual team composition, and its use reduced the time spent for team building by several hours each month."

Acknowledgements

The authors are thankful to the staff of B.L.G. at Gioia Tauro port; a special thanks is addressed to F. Scalise, G. Lucchese, and S. Papalia.

This work has been partially supported by the Regione Calabria and EU under POR Calabria FESR 2007-2013 within the PIA project of DLVSYSTEM.

References

Aickelin, U. and Dowsland, K. A. 2000. Exploiting problem structure in a genetic algorithm approach to a nurse rostering problem. *Journal of Scheduling 3,* 3, 139–153.

Al-Yakoob, S. and Sherali, H. 2007. Mixed-integer programming models for an employee scheduling problem with multiple shifts and work locations. *Annals of Operations Research 155,* 1, 119–142.

Alba, E. and Chicano, J. F. 2007. Software project management with GAs. *Information Sciences 177,* 11, 2380–2401.

Alfares, H. K. 2002. Optimum workforce scheduling under the (14, 21) days-off timetable. *JAMDS 6,* 3, 191–199.

Balduccini, M., Gelfond, M., Watson, R., and Nogueira, M. 2001. The USA-Advisor: A Case Study in Answer Set Planning. In *Proceedings of LPNMR '01,*. LNCS, vol. 2173. Springer Berlin / Heidelberg, 439–442.

Baral, C. 2003. *Knowledge Representation, Reasoning and Declarative Problem Solving.* Cambridge University Press.

Baral, C. and Gelfond, M. 2000. *Logic-based artificial intelligence.* Kluwer Academic Publishers, Chapter Reasoning agents in dynamic domains, 257–279.

Baral, C. and Uyan, C. 2001. Declarative Specification and Solution of Combinatorial Auctions Using Logic Programming. In *Logic Programming and Nonmotonic Reasoning.* LNCS, vol. 2173. Springer Berlin / Heidelberg, 186–199.

Bardadym, V. 1996. Computer-aided school and university timetabling: The new wave. In *Practice and Theory of Automated Timetabling.* LNCS, vol. 1153. Springer Berlin / Heidelberg, 22–45.

Bechtold, S. E., Brusco, M. J., and Showalter, M. J. 1991. A Comparative Evaluation of Labor Tour Scheduling Methods. *Decision Sciences 22,* 4, 683–699.

Billionnet, A. 1999. Integer programming to schedule a hierarchical workforce with variable demands. *European Journal of Operational Research 114,* 1, 105–114.

Burke, E. K. and Soubeiga, E. A Real-World Workforce Scheduling Problem in the Hospitality Industry: Theoretical Models and Algorithmic Methods.

Cerulli, R., Gaudioso, M., and Mautone, R. 1992. A class of manpower scheduling problems. *Mathematical Methods of Operations Research 36,* 1, 93–105.

Chiu, D. K. W., Cheung, S. C., and Leung, H.-f. 2005. A Multi-Agent Infrastructure for Mobile Workforce Management in a Service Oriented Enterprise. In *Proceedings of HICSS '05*. IEEE Computer Society, 85c–85c.

Dechter, R. 2004. *Constraint processing*. Morgan Kaufmann Publishers.

Eiter, T., Faber, W., Leone, N., and Pfeifer, G. 2000. *Logic-based artificial intelligence*. Kluwer Academic Publishers, Chapter Declarative problem-solving using the DLV system, 79–103.

Eitzen, G., Panton, D., and Mills, G. 2004. Multi-Skilled Workforce Optimisation. *Annals of Operations Research 127,* 1, 359–372.

Ernst, A. T., Jiang, H., Krishnamoorthy, M., and Sier, D. 2004. Staff scheduling and rostering: A review of applications, methods and models. *European Journal of Operational Research 153,* 1, 3–27.

Faber, W., Leone, N., and Pfeifer, G. 2004. Recursive Aggregates in Disjunctive Logic Programs: Semantics and Complexity. In *Logics in Artificial Intelligence*. LNCS, vol. 3229. Springer Berlin / Heidelberg, 200–212.

Franconi, E., Palma, A., Leone, N., Perri, S., and Scarcello, F. 2001. Census Data Repair: A Challenging Application of Disjunctive Logic Programming. In *Logic for Programming, Artificial Intelligence, and Reasoning*. LNCS, vol. 2250. Springer Berlin / Heidelberg, 561–578.

Friedrich, G. and Ivanchenko, V. 2008. Diagnosis from first principles for workflow executions. Tech. Rep. 2, Alpen-Adria-Universitt Klagenfurt.

Gebser, M., Liu, L., Namasivayam, G., Neumann, A., Schaub, T., and Truszczyński, M. 2007. The First Answer Set Programming System Competition. In *Logic Programming and Nonmonotonic Reasoning*. LNCS, vol. 4483. Springer Berlin / Heidelberg, 3–17.

Gelfond, M. and Leone, N. 2002. Logic programming and knowledge representation-the A-prolog perspective. *Artif. Intell. 138,* 1-2, 3–38.

Gelfond, M. and Lifschitz, V. 1991. Classical negation in logic programs and disjunctive databases. *New Generation Computing 9*, 365–385.

Grasso, G., Iiritano, S., Leone, N., and Ricca, F. 2009. Some DLV Applications for Knowledge Management. In *Logic Programming and Nonmonotonic Reasoning*. LNCS, vol. 5753. Springer Berlin / Heidelberg, 591–597.

Gresh, D. L., Connors, D. P., Fasano, J. P., and Wittrock, R. J. 2007. Applying supply chain optimization techniques to workforce planning problems. *IBM J. Res. Dev. 51,* 3, 251–261.

Hultberg, T. H. and Cardoso, D. M. 1997. The teacher assignment problem: A special case of the fixed charge transportation problem. *European Journal of Operational Research 101,* 3, 463–473.

Lau, H. C. 1996. On the complexity of manpower shift scheduling. *Computers & Operations Research 23,* 1, 93–102.

Lee, J. and Meng, Y. 2009. On Reductive Semantics of Aggregates in Answer Set Programming. In *Logic Programming and Nonmonotonic Reasoning*. LNCS, vol. 5753. Springer Berlin / Heidelberg, 182–195.

Leone, N., Greco, G., Ianni, G., Lio, V., Terracina, G., Eiter, T., Faber, W., Fink, M., Gottlob, G., Rosati, R., Lembo, D., Lenzerini, M., Ruzzi, M., Kalka,

E., Nowicki, B., and Staniszkis, W. 2005. The INFOMIX system for advanced integration of incomplete and inconsistent data. In *Proceedings of SIGMOD '05, Baltimore, Maryland*. ACM, New York, NY, USA, 915–917.

Leone, N., Pfeifer, G., Faber, W., Eiter, T., Gottlob, G., Perri, S., and Scarcello, F. 2006. The DLV system for knowledge representation and reasoning. *ACM Trans. Comput. Logic 7,* 3, 499–562.

Lesaint, D., Voudouris, C., Azarmi, N., Alletson, I., and Laithwaite, B. 2003. Field workforce scheduling. *BT Technology Journal 21,* 4, 23–26.

Naveh, Y., Richter, Y., Altshuler, Y., Gresh, D. L., and Connors, D. P. 2007. Workforce optimization: Identification and assignment of professional workers using constraint programming. *IBM Journal of Research and Development 51,* 3.4, 263–279.

Nogueira, M., Balduccini, M., Gelfond, M., Watson, R., and Barry, M. 2001. An A-Prolog Decision Support System for the Space Shuttle. In *Practical Aspects of Declarative Languages*. LNCS, vol. 1990. Springer Berlin / Heidelberg, 169–183.

Ricca, F., Gallucci, L., Schindlauer, R., Dell'Armi, T., Grasso, G., and Leone, N. 2009. OntoDLV: An ASP-based System for Enterprise Ontologies. *J Logic Computation 19,* 4, 643–670.

Rossi, F. 2000. Constraint (Logic) Programming: A Survey on Research and Applications. In *New Trends in Constraints*. LNCS. Springer Berlin / Heidelberg, 40–74.

Sun, M., Aronson, J. E., McKeown, P. G., and Drinka, D. 1998. A tabu search heuristic procedure for the fixed charge transportation problem. *European Journal of Operational Research 106,* 2-3, 441–456.

Tien, J. M. and Kamiyama, A. 1982. On manpower scheduling algorithms. *SIAM Review 24,* 3, 275–287.

Vacca, I., Bierlaire, M., and Salani, M. 2007. Optimization at Container Terminals: Status, Trends and Perspectives. Swiss Transport Research Conference, Ascona, Switzerland, September 14, 2007.

Wren, A. and Wren, D. O. 1995. A genetic algorithm for public transport driver scheduling. *Computers & Operations Research 22,* 1, 101–110.

Yang, R. 1996. Solving a Workforce Management Problem with Constraint Programming. In *Proceedings of PACT '96*. The Practical Application Company Ltd, 373–387.